

+

1/29

FIG. 1
PRIOR ART

```
1 c:\collections
2     notes.txt
3     myletter.doc
4     c-myhomepage
5
6     s
7         homepage.html
8         myphoto.jpg
```

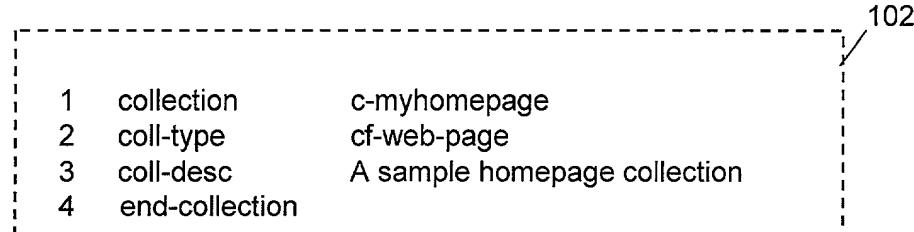
FIG. 2

```
1 c:\collections
2     notes.txt
3     myletter.doc
4     c-myhomepage
5     cspec
6     s
7         homepage.html
8         myphoto.jpg
```



FIG. 3

```
1 collection      c-myhomepage
2 coll-type       cf-web-page
3 coll-desc       A sample homepage collection
4 end-collection
```



+

FIG. 4

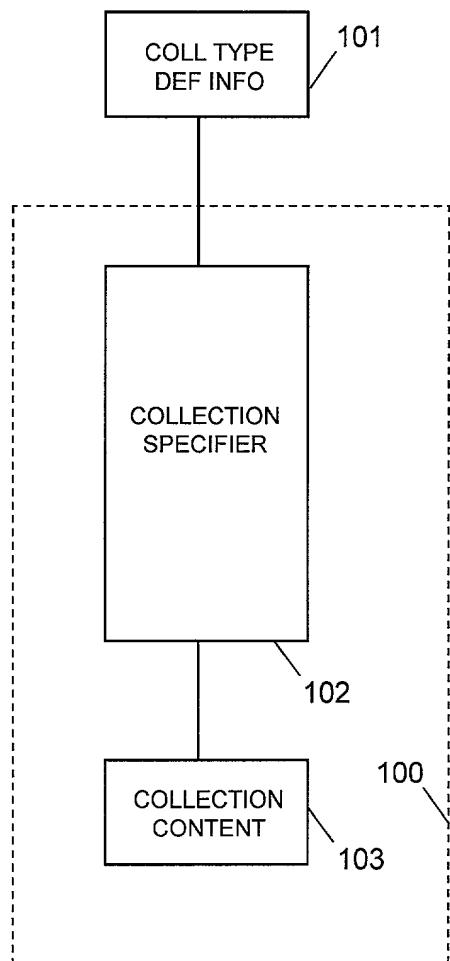


FIG. 5

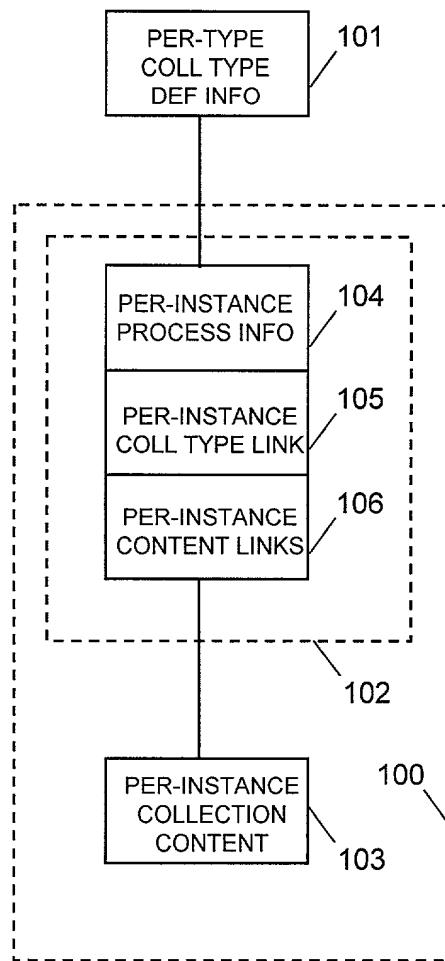


FIG. 6

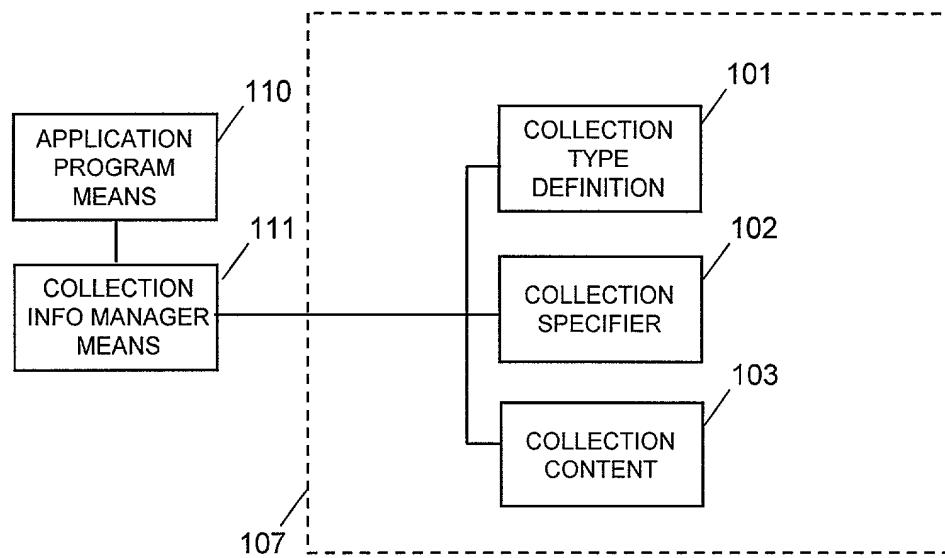


FIG. 7

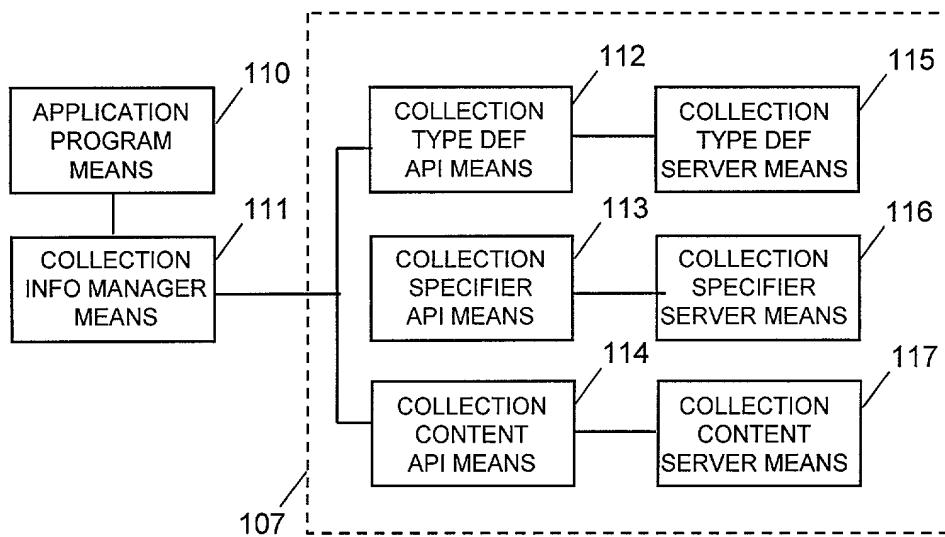


FIG. 8

```
1 /* collection data structure */
2 collection-info {
3     + specifier_info
4         + coll-type-indicator
5         + other specifier information ...
6     + content_info
7         + content_location_info ...
8         + content_members ...
9         + other content information...
10    + other collection structure information...
11 }
```

FIG. 9

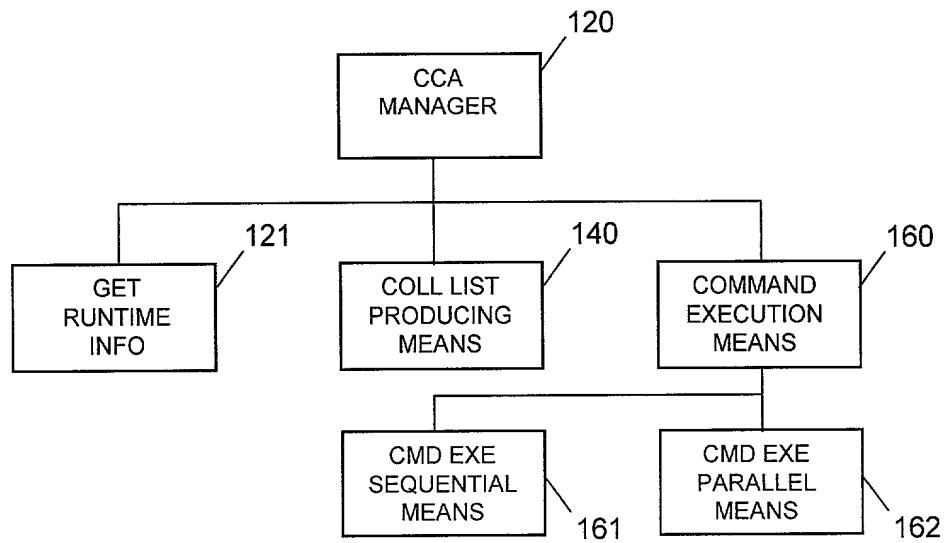
```
1 /* collection type definition data structure */
2 collection-type-definition-info {
3     + coll-type-name
4     + collection internal structure info ...
5     + collection content location info ...
6     + collection content type recognition info ...
7     + other collection type definition information...
8 }
```

FIG. 10

<u>KEY</u>	<u>VALUE</u>
1 /* collection type internal structure definitions */	
2 dir_source_files	\s
3 dir_doc_files	\doc
4 /* content location definitions (per-type content links) */	
5 content_subtree_http	http://host.com/some/dir/name
6 content_subtree_ftp	ftp://host.com/some/dir/name
7 content_subtree_nfs	/some/local/directory/name
8 /* content type recognition definitions */	
9 content_policy	subtree_below_cspec_file
10 content_file_type	.c file_cpp
11 content_file_type	.c file_c
12 content_file_type	.h file_c_include
13 content_file_type	.doc file_ms_word
14 content_file_type	.html file_html
15 content_file_type	.xls file_ms_excel
16 /* collection processing definitions */	
17 compile_c_files	yes
18 compiler_windows	vc++
19 compiler_unix	gcc
20 build platforms	Win98, Win2000, linux
21 process files	compile link
22 link libraries	stdio math sock
23 /* results dispatching definitions */	
24 results_ftp_host	ftp.output.com
25 results_ftp_dir	c:\ftphome\collection\results

6/29

FIG. 11



7/29

FIG. 12

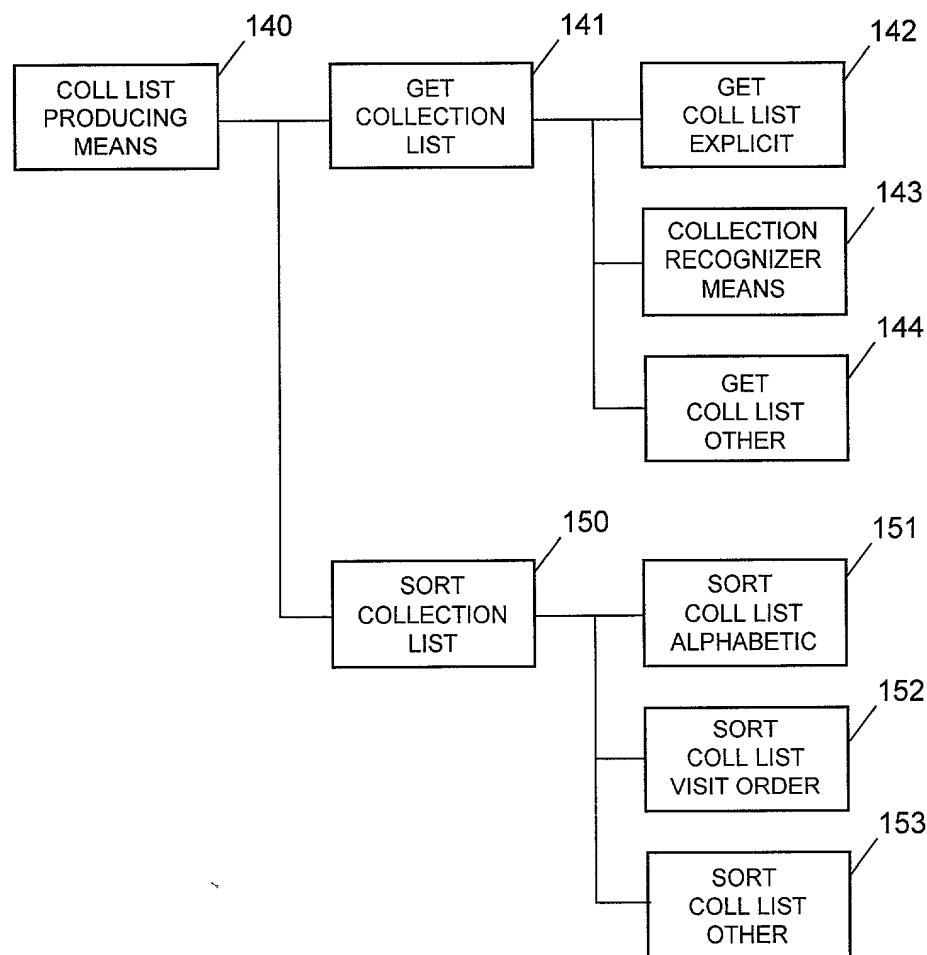
- 1 /* simplified cca algorithm */
- 2 Call get runtime information to obtain a list of commands to apply
- 3 Call collection list producing means to obtain a list of target collections for command application
- 4 Call command execution means to apply commands to target collections

FIG. 13

- 1 /* runtime information data structure */
- 2 runtime-info {
- 3 + invocation options
- 4 + runtime environment information
- 5 + commands to execute on recognized collections
- 6 + collection recognition criteria
- 7 ...
- 8 }

8/29

FIG. 14



+

9/29

FIG. 15

- 1 /* simplified algorithm for list producing means */
 - 2 Build data structures

 - 3 /* Obtain list of target collections for command application */
 - 4 - add explicit invocation-provided collections to list
 - 5 - add collection-recognizer-provided collections to list
 - 6 - add collections from other sources to list

 - 7 /* Build sorted-colls data structures */
 - 8 - create list of target collections sorted alphabetically
 - 9 - create list of target collections sorted by visit order
 - 10 - create list of target collections sorted by other means

 - 11 Return completed coll-list-info data structure to caller
- +

10/29

FIG. 16

```
1 /* list of target collections for command application */
2 target-coll-list {
3     + list of collection-structures (eg FIG 8)
4 }
```

FIG. 17

```
1 /* structure for holding sorted lists of collections */
2 sorted-colls {
3     + sort-type = ALPHA, VISIT_ORDER, ...
4     + sorted-collection-set {
5         + coll-1 in sort order
6         + coll-2 in sort order
7         + ...
8 }
```

FIG. 18

```
1 /* structure for holding collection list info */
2 coll-list-prod-info {
3     + list of collection structures (target-coll-list)
4     + list of collection type definition structures (FIG 9)
5     + list of sorted-colls structures
6     + collection recognition info ...
7 }
```

+

11/29

FIG. 19

```
1 c:\collections
2     programs
3         helloworld
4             c-hello-library
5             c-hello
6
7     c-myprogram
8
9     parts
10        c-include-files
11        c-library-one
12        c-library-two
13
14     webstuff
15         c-myhomepage
16
17     c-myphotos
```

FIG. 20

```
0 /* File colls-fig-20.txt holds list of collections for tree FIG 19 */
1 c:\collections\programs\helloworld\c-hello-library
2 c:\collections\programs\helloworld\c-hello
3 c:\collections\c-myprogram
4 c:\collections\parts\c-include-files
5 c:\collections\parts\c-library-one
6 c:\collections\parts\c-library-two
7 c:\collections\webstuff\c-myhomepage
8 c:\collections\c-myphotos
```

12/29

FIG. 21

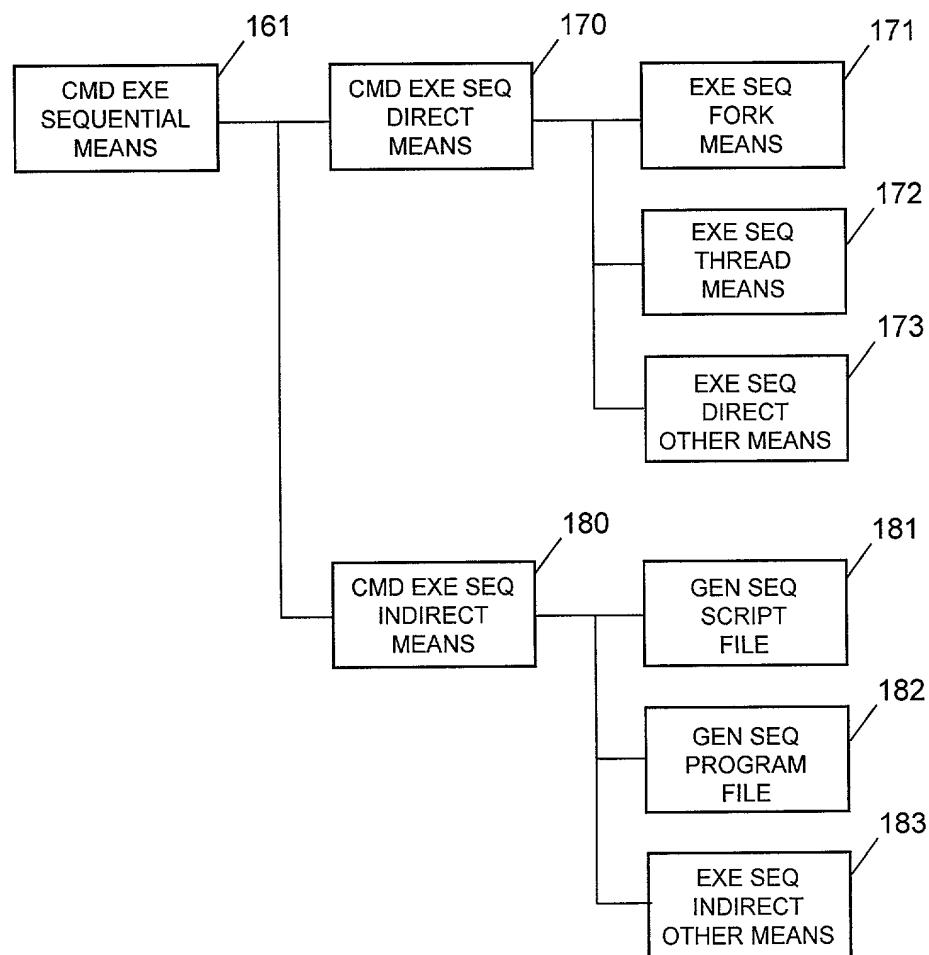


FIG. 22

```
1 /* simplified algorithm for execute sequential direct means */
2 Build data structures

3 /* walk list of target collections */
4 For each coll in list of one-coll-cmd-exe structures
5   - change working directory to desired execution directory

6 /* execute direct commands on each target collection */
7   - for each command in list of cmd-exe-status structures
8     - execute the command using a subordinate helper module
9     - record command status and errors
10    - if command status = FAILS continue with next collection
11    - continue with next command in list of commands

12 /* clean up and return status */
13 Change working directory back to original invocation directory
14 Return overall execution status to caller
```

14/29

FIG. 23

```
1 /* command and status info for 1 command */
2 cmd-exe-status {
3     + command to execute
4     + return code status of execution attempt
5     + other error information
6     + ...
7 }
```

FIG. 24

```
1 /* N commands and status info for 1 target collection */
2 one-coll-cmd-exe {
3     + target collection for commands
4     + list of cmd-exe-status structures
5     + ...
6 }
```

FIG. 25

```
1 /* commands and status info for all target collections */
2 all-coll-cmd-exe {
3     + list of coll-cmd-exe structures
4     + execution-type = DIRECT, INDIRECT
5     + exe-direct-method = FORK, THREAD, ...
6     + exe-indirect-method = BATCH, PERL, ...
7 }
```

15/29

FIG. 26

```
1 if not ()==(%)1 goto goodargs
2 echo Usage: doinseq command1 c2 c3 ... c5
3 echo          doinseq make all
4 echo          doinseq dir *.c
5 goto quit

6 :goodargs

7 cd c:\collections\c-myphotos\win98.plt
8 call %1 %2 %3 %4 %5 %6 %7 %8 %9
9 cd c:\collections

10 cd c:\collections\parts\c-library-one\win98.plt
11 call %1 %2 %3 %4 %5 %6 %7 %8 %9
12 cd c:\collections

13 cd c:\collections\programs\helloworld\c-hello\win98.plt
14 call %1 %2 %3 %4 %5 %6 %7 %8 %9
15 cd c:\collections
16 ...
17 :quit
```

FIG. 27

```
1 C:\> cca doinseq --explicit-colls colls-fig-20.txt --platform win98.plt
2 C:\> doinseq <a-command-to-execute>
3 C:\> doinseq ls
4 C:\> doinseq make all
```

FIG. 28

- 1 /* simplified algorithm for execute sequential indirect */
- 2 Build data structures
- 3 Generate a batch file template framework to hold commands
- 4 /* walk list of collections and generate commands into the batch file */
- 5 For each coll in list of coll-cmd-exe structures
- 6 - emit command to change working directory into
desired execution directory
- 7 - emit script file argument variables to hold commands that
are passed in to the script file for execution
- 8 - emit command to change back to original working directory
- 9 - continue with next collection in list of collections
- 10 Return batch file execution status

FIG. 29

```
1 /* rec-coll recognized-collections data structure */

2 rec-coll {
3     + rec-coll-list

4     + coll-structure-1
5         + cspec_info ...
6         + ctype_def_info ...
7         + ccontent_info ...
8         + other_coll_info

9     + coll-structure-2
10    + cspec_info ...
11    + ctype_def_info ...
12    + ccontent_info ...
13    + other_coll_info

14    + coll-structure-3
15    ...

16    + other collection recognition info
17 }
```

FIG. 30

```
1 c:\collections
2     programs
3         helloworld

4         c-hello           ctype = cf-program
5             default vo=100
6         c-hello-library   ctype = cf-library
7             default vo=50

6         c-myprogram       ctype = cf-program
7             default vo=100

7         parts
8             c-include-files  ctype = cf-includes
9                 default vo=10
10            c-library-one   ctype = cf-library
11                default vo=10
12            c-library-two   ctype = cf-library
13                EXPLICIT vo=49

11        webstuff
12            c-myhomepage    ctype = cf-doc-html
13                default vo=100

13        c-myphotos        ctype = cf-web-page
14            default vo=100
```



19/29

FIG. 31

	Collection Type Name	Visit Order Ranking
1	cf-initial	10
2	cf-library	50
3	cf-program	100
4	cf-web-page	100
5	cf-doc-sgml	100
6	cf-doc-html	100

FIG. 32

1	collection	c-library-two
2	coll_type	cf-library
3	coll_desc	A library with explicit visit order
4	coll-visit-order	49
5	end-collection	

FIG. 33

1	c-hello	100
2	c-hello-library	50
3	c-myprogram	100
4	c-library-one	50
5	c-library-two	49
6	c-include-files	10
7	c-myhomepage	100
8	c-myphotos	100



FIG. 34

- 1 /* simplified visit order algorithm */
- 2 Receive unsorted list of collections
- 3 Obtain numeric visit order values for each collection in list
- 4 Sort the list of collections according to execution visit order
- 5 Write sorted information to sorted-colls data structure (FIG 17)
- 6 Return sorted-colls data structure to calling module

FIG. 35

1	c-include-files	10
2	c-library-two	49
3	c-library-one	50
4	c-hello-library	50
5	c-hello	100
6	c-myphotos	100
7	c-myhomepage	100
8	c-myprogram	100

21/29

FIG. 36

```
1 if not ()==(%)1 goto goodargs
2 echo Usage: doinseq command1 c2 c3 ... c5
3 echo           doinseq make all
4 echo           doinseq dir *.c
5 goto quit

6 :goodargs

7 cd c:\collections\parts\c-include-files\win98.plt (vo=10)
8 call %1 %2 %3 %4 %5 %6 %7 %8 %9
9 cd c:\collections

10 cd c:\collections\parts\c-library-two\win98.plt (vo=49)
11 call %1 %2 %3 %4 %5 %6 %7 %8 %9
12 cd c:\collections

13 cd c:\collections\parts\c-library-one\win98.plt (vo=50)
14 call %1 %2 %3 %4 %5 %6 %7 %8 %9
15 cd c:\collections

16 cd c:\collections\programs\helloworld\c-hello-library\win98.plt
17 call %1 %2 %3 %4 %5 %6 %7 %8 %9 (vo=50)
18 cd c:\collections

19 cd c:\collections\programs\helloworld\c-hello\win98.plt
20 call %1 %2 %3 %4 %5 %6 %7 %8 %9 (vo=100)
21 cd c:\collections
22 ...

23 :quit
```



22/29

FIG. 37

	Visit Order Set Name	Visit Order Definition File
1	vo-software	vo-software.def
2	vo-doc	vo-doc.def
3	vo-xxx-name	vo-xxx-name.def

FIG. 38

0	vo-software.def:	
1	cf-initial	10
2	cf-library	50
3	cf-program	100
4	cf-web-page	100
5	cf-doc-sgml	100
6	cf-doc-html	100

FIG. 39

0	vo-doc.def:	
1	cf-doc-sgml	10
2	cf-doc-html	10
3	cf-doc-indexes	20





23/29

FIG. 40

1	collection	c-program-doc
2	coll_type	cf-doc-sgml
3	coll_desc	A doc with multiple explicit visit orders
4	coll-visit-order	vo-software 49
5	coll-visit-order	vo-doc 10
5	end-collection	

FIG. 41

```
1 /* simplified algorithm for calculating parallel execution groups */
2 Obtain list of target collections, sorted into proper visit order
3 Obtain physical parallelism limit = phys_par_limit
4 Obtain administrative parallelism limit = admin_limit

5 /* calculate problem parallelism limit */
6 set problem_par_limit = 1
7 For each unique visit order in list of target collections,
8   - count number of collections with current visit order value
9   - if current_count > problem_par_limit,
10     set problem_par_limit = current_count

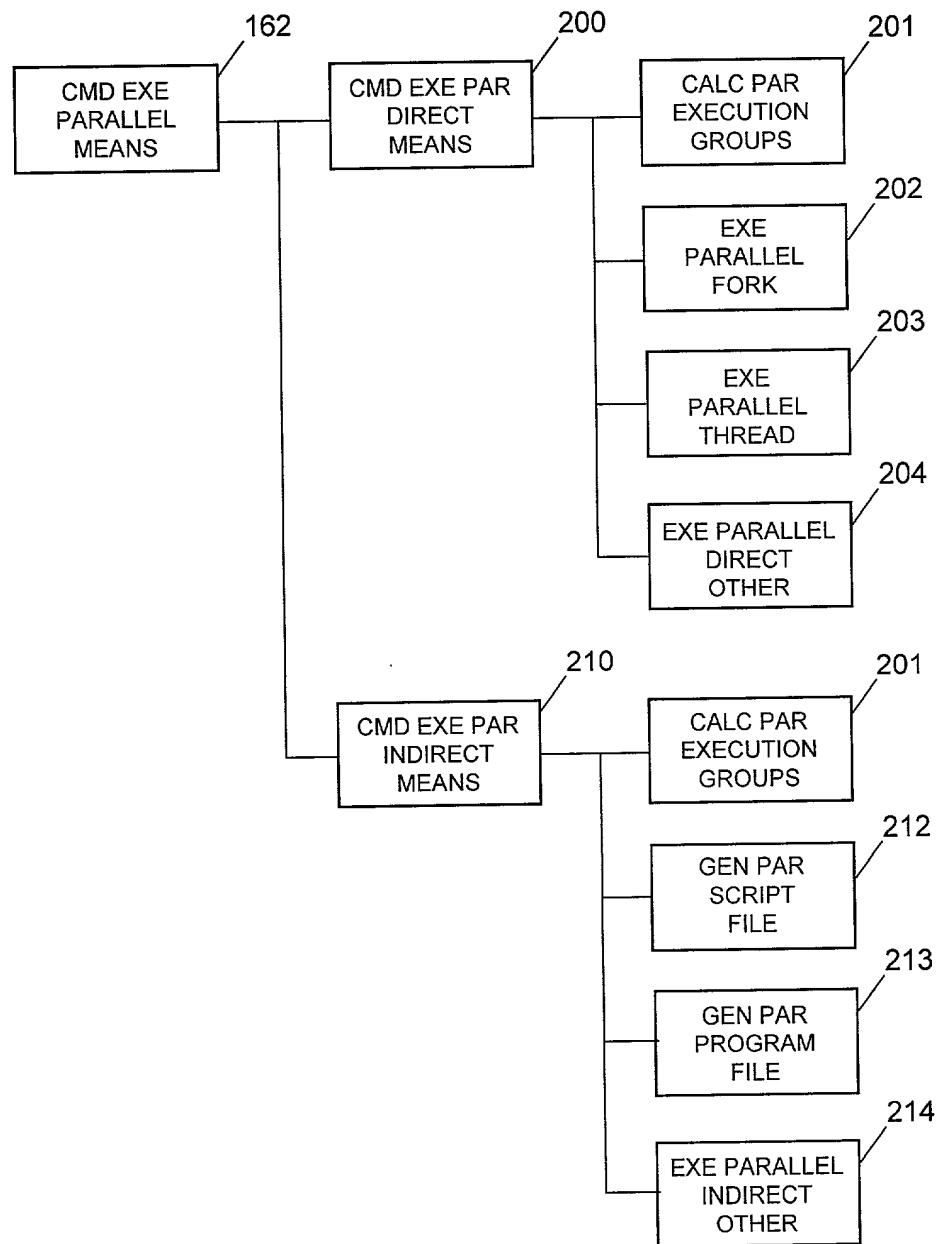
11 /* calc min of problem, physical, and admin parallelism limits */
12 useful_par_limit = min (problem_par_limit, physical_par_limit,
                           admin_par_limit)

13 /* calc parallel execution groups using useful_par_limit */
14 For each unique visit order in list of target collections,
15   - create a new parallel execution group
16   containing all collections that match the current visit order
17   - if group size is > useful_par_limit,
18     split group into smaller groups
19     until no groups exceed useful_par_limit
20   - continue with next unique visit order in list of target collections

21 Return parallel execution ordering and limits to caller
```



FIG.42



25/29

FIG. 43

```
1 /* simplified algorithm for execute parallel direct */
2 Build data structures

3 /* calculate parallel execution groups */
4 Calculate parallel execution groups per algorithm FIG 42

5 /* execute collections in each parallel group in parallel */
6 For each parallel execution group, in proper parallel exe order {
7   - Execute in parallel {
8     - For each collection in the parallel exe group
9       - change working directory to desired execution directory

10  /* execute commands for one collection */
11    - for each command in list of cmd-exe-status structures
12      - execute the command
13      - record command status and errors
14      - if command status = FAILS continue with next collection
15      - continue with next command in list of commands

16    - Wait for all collections in the group to finish
17  } /* end of parallel section beginning on Line 7
18 } /* end of parallel execution group list traversal beginning on Line 6

19 Return overall execution status information to caller
```

FIG. 44

```
1 /* collections for 1 parallel execution group */
2 parallel-exe-group {
3   + par-exe-rank = 1, 2, 3, ...
4   + par-exe-group-coll-list {
5     + collection-1
6     + collection-2
7     +
8 }
```

FIG. 45

```

1 /* collections for all parallel execution groups */
2 cmd-exe-parallel {

3     + list of coll-cmd-exe structures
4     + list of parallel-exe-group structures
5     + exe-type = DIRECT, INDIRECT
6     + exe-method-direct = FORK, THREAD, ...
7     + exe-method-indirect = PERL, PROGRAM,
8 }
```

FIG. 46

1	time=0, vo=10	c-include-files
2	time=1, vo=49	c-library-two
3	time=2, vo=50	c-library-one
		c-hello-library
5	time=3, vo=100	c-hello
6		c-myprogram
7		c-myhomepage
8		c-myphotos

FIG. 47

```
1 /* simplified algorithm for execute parallel indirect */

2 Build data structures

3 /* calculate parallel execution groups */

4 Calculate parallel execution groups per algorithm FIG 42

5 /* execute collections in each parallel group in parallel */
6 For each parallel execution group, in proper parallel exe order {
7   - Emit commands for executing in parallel {
8     - For each collection in the parallel exe group
9       - emit cd command to change to desired execution directory

10 /* emit execution commands for one collection */
11   - for each command in list of cmd-exe-status structures
12     - emit syntax for the command to be executed in parallel
13     - emit syntax to record command status and errors
14     - continue with next command in list of commands

15   - Emit syntax to wait for all collections in the group to finish
16 } /* end of parallel section beginning on Line 7
17 */ /* end of parallel execution group list traversal beginning on Line 6

18 Return overall execution status information to caller
```

FIG. 48

```

1 #!/bin/sh
2 if [ $# -lt 1 ] ; then
3     echo "Usage: doinparallel command-1 c-2 c-3 ... c-N"
4     echo "          doinparallel copy file1 file2"
5     echo "          doinparallel make all"
6     exit 1
7 fi

8 # at time 0, apply parallel execution group #1
9 cd /collections/parts/c-include-files/linux.plt
10 $@
11 cd /collections

12 ... # execution group #2 omitted to save space

13 # at time 2, apply commands parallel execution group #3
14 cd /collections/parts/c-library-one/linux.plt
15 $@ &
16 cd /collections/programs/helloworld/c-hello-library/linux.plt
17 $@ &
18 # wait for all parallel jobs to complete
19 wait
20 cd /collections

21 # at time 3, apply commands to parallel execution group #4
22 cd /collections/programs/helloworld/c-hello/linux.plt
23 $@ &
24 cd /collections/c-myprogram/linux.plt
25 $@ &
26 cd /collections/webstuff/c-myhomepage/linux.plt
27 $@ &
28 cd /collections/c-myphotos/linux.plt
29 $@ &
30 # wait for all parallel jobs to complete
31 wait
32 cd /collections

33 exit 0

```

+

29/29

FIG. 49

```
1 c:\collections
2     programs
3         helloworld
4             c-hello
5                 cspec
6                     s...
7                         win98.plt...
8                             gnulinux.plt...

9     c-myprogram
10        cspec
11            s...
12                win98.plt...
13                    gnulinux.plt...

14    parts
15        c-include-files
16            cspec
17                s...
18                    win98.plt...
19                        gnulinux.plt...
```

File Edit View Insert Format Tools Help

+

FIG. 50

- 1 Visit all s dirs beside cspec files, non-recursively
- 2 Visit all immediate child dirs, recursively
- 3 Format files in all s dirs (visit s dirs)
- 4 Modify all cspec files (visit root dirs of collections)
- 5 Delete all collections (visit parent dirs of collections)
- 6 Clean up win98.plt dirs (visit win98.plt dirs)
- 7 Delete all plt dirs (visit root dirs of collections)